

September/October 1986 Edition

RE RUN

RUN Programs on Disk

For the
C-64 and C-128*



A SPECIAL LOOK AT SMALL BUSINESS:

- Applications
- Buyer's Guide
- CAD Programs Reviewed
- Easy-to-Use Personal Ledger

INTRODUCING

MEGA BASIC, a New Column
Of Useful Typing Programs

MEET

Computing guru
Jim Butterfield

PLUS:

An Improved Puzzle Program
For Endless Hours of Fun



Plus: Bonus Program!



RUN'S EXTENDED BASIC
Add More Power to Your C-64

64 NOTEPAD
A Simple Desktop Accessory

TIPS FROM A PRO
On Taking Screen Shots

TURTLE GRAPHICS TUTOR
FOR CHILDREN

Plus:
CES NEW PRODUCT ROUNDUP

ORGANIZE YOUR HOME FINANCES

- Guide to Available Software
- Easy-to-Use Type-In Program



*128 mode
programs included

Introduction

September-October '86 ReRUN

Welcome to the September-October edition of ReRUN. September and October were theme issues for *RUN* magazine, covering personal finance and small-business applications, so you'll find several programs on this disk to help you in these areas. These issues also contained programs to help you use your Commodore for education, programming and graphics.

Let's begin with business and finance. The Loan Arranger is a valuable finance aid for both 64 and 128 users. With it, you can figure out your monthly loan payments, loan balances and amortization schedules.

64 Personal Ledger helps you keep track of savings accounts and transactions involved with running a small business. Book-keeping tasks become easier with this computerized ledger.

Sign Maker lets you create banners and signs with characters large enough for people to read from across the room or

across the street. Two formats are available: sideways printing for banners up to 40 characters high and standard printing for smaller signs.

Make your business presentations clearer with Making a Pointer, a screen-oriented visual aid for focusing attention on specific aspects of your monitor displays. This is perfect for use with large-size screens, which many schools, small businesses and user groups employ.

Speaking of screen displays, we have several graphics programs for the whole family. For children, take a look at Turtle-Tutor for Tykes, a first step into LOGO programming on the C-64. For less structured fun, try running Double Vision, which gives your kids a treat as they create colorful, mirrored designs.

For experienced programmers, check out RUN Basic. The version on this disk has already been combined with Basic 4.5, so all you need do is boot RUN

Basic to have all of the commands from these two major Basic extensions. This gives you extra commands for programming disk operations, sprite graphics, music, windows, turtle graphics and subroutines.

Specifically aimed at simplifying high-resolution drawing is High-Resolution Revolution. Once you run this program, you'll have eight powerful commands for drawing on your C-64.

Mega-Magic is *RUN*'s newest monthly column, featuring some of the larger Magic submissions that are sent to *RUN*. In October, we published Scroller, a machine language routine that lets you scroll your C-64's screen horizontally.

RUN's Basically Speaking columns have provided much information for those who want to learn how to program in Basic. Instant Data Statements, from September, lets 64 and 128 owners convert machine language programs into data statements for use within their own Basic programs. Chain Your Programs,

from October, is a utility for programmers to use when their program creation exceeds the computer's available memory.

For keeping track of important notes and numbers, we have Programmers, Take Note! This computerized notepad resides in your computer's memory while you program or run other programs. Just press a key when you want the notepad to appear on screen.

For educational fun, As the Word Turns is bound to keep you and your family intrigued as you try to find the words hidden in this puzzle.

Lastly, to put you in the mood for the Halloween season, we have a bonus program, Halloween Story. This program will raise eyebrows when you run it on All Saint's Eve.

That's all for this edition of ReRUN. Have fun!

Margaret Morabito
Technical Manager
RUN magazine

How To Load

Loading from Menu

To get started, C-64 users should type `LOAD "MENU 64",8` and press the return key. When you get the Ready prompt, the menu is loaded and you should type `RUN` to see a list of the programs on your disk.

Loading from Keyboard

If you do not wish to use the menu program, follow these instructions.

C-64:

To load a C-64 program written in Basic, type:

`LOAD "DISK FILENAME",8`

and then press the return key. The drive will whirl while the screen prints `LOADING` and then `READY`, with a flashing cursor beneath. Type `RUN` and press the return key. The program will then start running.

To load a C-64 program written in machine language (ML), type:

`LOAD "DISK FILENAME",8,1`

C-128:

All C-64 programs can be run on the C-128 as long as your computer is in C-64 mode.

All C-128 programs are clearly labeled on the directory page. Your C-128 *must* be in C-128 mode to run these programs.

To load a C-128-mode program, press the F2 key, type the disk filename and then press the return key. When the program has loaded, type `RUN`.

Making Copies of ReRUN Disks

Many of the programs on your ReRUN disk have routines that require you to have a separate disk onto which the program writes or saves subfiles. In order for you to use these programs, you will first have to make a copy of the original program onto another disk that has enough free space on it to hold these newly written subfiles.

If the program is written in Basic, it is simple to make a copy of the program. Just load the program into your computer following the procedures outlined above, and then save the program back onto a separate disk that has plenty of free space for extra files.

If the program is written in ML, copying is not so simple. You cannot simply load and save an ML program. In this case, you'll need to use a disk-backup utility program, such as the one on your Commodore Test Demo disk.

ReRUN Staff

Technical Manager/Editor: *Margaret Morabito*

Technical Editor: *Tim Walsh*

Managing Editor/Production: *Swain Pratt*

Copy Editor: *Peg LePage*

Proofreader: *Harold Bjornsen*

Design and Layout: *Karla M. Whitney*

Typesetting: *Doreen Means, Beth Krommes, Ken Sutcliffe*

Directory

Page	Article	Disk Filename	File Type
		MENU 64	BASIC
1	The Loan Arranger	LOAN ARRANGER (64/128)	BASIC
4	Turtle-Tutor for Tykes	TURTLE TUTOR	BASIC
7	RUN Basic	RUN BASIC 1.0	ML
		DEMO 1	BASIC
		DEMO 2	BASIC
		DEMO 3	BASIC
		DEMO 4	BASIC
		DEMO 5	BASIC
		DEMO 6	BASIC
15	Programmers, Take Note!	NOTEPAD	BASIC
17	Sign Maker	SIGN MAKER	BASIC
18	Instant Data Statements	DATAMAKER 64	BASIC
		DATAMAKER 128	BASIC
21	64 Personal Ledger	BALANCE SHEET	BASIC
25	High-Res Revolution	64/GRAFIX	BASIC
		GRAFIX DEMO 1	BASIC
		GRAFIX DEMO 2	BASIC
28	As the Word Turns	FIND THE WORD	BASIC
30	Making a Pointer	DEMO POINTER	BASIC
32	Double Vision	MIRROR	BASIC
33	Chain Your Programs	OVERLAY.BOOT	BASIC
		MAIN	BASIC
		SUB1	BASIC
38	Mega-Magic	*SCROLLER	BASIC
		RANDOM SCROLLER	BASIC
39	£ Halloween Story	HALLOWEEN	BASIC
		PUMPKIN	ML

NOTE: Do not load indented files as stand-alone programs!

* Be sure to place a different disk in your drive before running SCROLLER.
This program writes a machine language program to disk.

£ Bonus program!

The Loan Arranger

By Jaap Kroes

RUN It Right

C-64; C-128 (in 40-column mode)

Printer optional

How would you like an amortization schedule of each outstanding loan you have, including the mortgage? By having such a schedule, you can easily calculate for tax-deduction purposes the amount of interest you paid during the preceding year.

The Loan Calculator and Amortization program for the C-64 and C-128 will do all that for you. It's easy to use, requiring only input to the questions asked on the screen. And if you want a hard copy of the amortization schedule, just answer *Y* when prompted for it, and you'll get a neat, easy-to-read printout.

USING THE PROGRAM

After you run the program, a menu will appear.

Option 1, Monthly Payments, consists of three questions to determine your monthly payments. You'll be prompted to enter your

loan principal, the interest rate and the number of months in the repayment schedule. This is a handy feature, since you may wish to see what effect different interest rates would have on your monthly payment.

Option 2, Loan Balance, begins by prompting you to enter the amount of your current loan. It then asks you for the amount of your monthly payments and how many you've made, and finally for the interest rate. The remaining balance is instantly displayed, and you are asked if you want to run another. An *N* answer returns you to the main menu.

The most interesting part of the program is option 3, Amortization, which will list the figures to your screen or printer.

Again you are asked to input the amount of the loan, the interest rate and the duration of the loan in months. You are then asked if you want a hard copy. If you answer *N*, the amortization schedule of your loan is printed to the screen. The top three lines of the screen consist of the column headings. The figures scroll

LOAN MENU

- 1) MONTHLY PAYMENT
- 2) LOAN BALANCE (PAYOFF)
- 3) AMORTIZE A LOAN
- 4) END PROGRAM

through the rest of the normal screen.

After the program loops through the formulas—one loop per month—you are informed of the total amount of your payments and the cost of the loan. The cost of the loan is nothing more than the total of payments minus the principal amount of the loan.

Please note that on occasion, when working with large loans, the amounts may be off by a few cents. This is due to the method of rounding, which doesn't always go to the nearest penny. However, the program has been exceptionally accurate in most situations.

If you answer Y when asked if you want a hard copy, you'll get a printout of the amortization schedule. The routine should work fine with your Commodore printer as well as with third-party printers. Tabbing is accomplished in a rather crude but effective way, using

```
PRINT#4,""TAB(n)X$
```

where n is the number of spaces to tab.

Page advance is automatic by means of counter K. When this counter reaches 42 lines, the program prints a message at the bottom of the page and increases the page number by 1. A form-feed is

M O N T H L Y P A Y M E N T

- 1) LOAN PRINCIPLE ? 12500
- 2) LOAN INTEREST ? 2.9
- 3) LOAN DURATION (MONTHS)? 36
- 4) MONTHLY PAYMENT = \$ 362.96

ANOTHER Y/N

sent to the printer, and the counter K is reset to 0. The next page number appears at the top of the next page and is followed by the column headings.

When you wish to exit the program, select option 4. A final message is printed to the screen, and the program ends.

All in all, you should find the program to be a very accurate

and useful addition to your financial program library. And it should help you make some informed decisions about borrowing money prior to your actual shopping. The amortization schedule makes tax time a little easier, since you don't have to rely on lenders to provide you with interest-paid information. 

Turtle-Tutor For Tykes

By Peter Crosby

RUN It Right

C64

Children under six or seven are fascinated by computers, but limited in what they can create with them, since they can't read or handle detail well enough to program. I wrote Elmer the Turtle, an introductory turtle-graphics program, for my own children so they could start to program, and I've found that it can be useful fun for beginning adults, too.

Elmer is a pen-wielding "turtle" who moves about the screen and draws according to a list of instructions you create by copying choices from a menu. The programming is done with only two screens, the second following the first automatically. There's no switching from menu to menu as in more advanced programs like Logo. Eleven simple statements are sufficient to put Elmer through reasonably complex ma-

neuvers, and four rudimentary editing commands enable you to arrange the program listing.

The program is reasonably crash-proof. If you type in garbage, Elmer just says he doesn't understand and would you please try again. If it does crash, you can usually restart it without losing anything by typing GOTO 700.

The statement list for Elmer is limited to 36 lines so that they'll all fit on one screen. Obviously, after a while you'll want more room. That's when you move on to Logo or regular turtle graphics. Since Elmer's vocabulary and procedures do carry over, you'll have a valuable head start.

TALKING TO ELMER

As I mentioned, there can be up to 36 instructions in a list. Each instruction has a line number. After a couple of introductory screens that explain what the program is about, a display appears with an empty list of all

RIGHT
UP
DOWN
INSERT
LEFT
PEN UP
PEN DOWN
DELETE
CHAR
COLOR
TEXT
BACK TO LINE #
STOP

Table 1. Commands for directing Elmer and for editing the instruction list.

the line numbers and, at the bottom, a menu of three choices.

You can go to a line number you specify to write in an instruction; you can type E to make Elmer follow the instructions you've already written; or you can type NEW to clear your list and start afresh. Of course, when you're just beginning to play, only the first choice, writing instructions, is viable. So, type in the line number you want—probably 1—and hit the return key. The next screen displays all the possible instructions for making Elmer walk and draw, and for editing the list. (See Table 1.)

Choose a direction for Elmer to walk by typing the appropriate word, then tell him how many steps he should take in that direction. You must include the number—he won't understand otherwise. When you press the return key again, your instruction will appear at the specified line number in the list.

To make Elmer draw, tell him PEN DOWN. He'll place his pen on the screen and draw a trail behind him. When you say PEN UP, he'll lift the pen and leave no mark at all. The drawing instructions do not need following numbers.

Elmer usually uses an asterisk for drawing his trail, because he thinks it looks like a turtle. However, he'll draw with any other character you choose. Just type the instruction CHAR, followed by a space and the character you want. For example, CHAR E makes him leave a trail of Es. You can request any letter, number or punctuation mark on the keyboard except the Commodore graphics symbols.

Elmer can draw in 16 different colors. To change color, type COLOR, a space and the number of your choice, 0 to 15. If you have a monochrome monitor, you can choose from seven shades.

Even though he's just a silly old turtle, Elmer can write mes-

sages if you tell him what to say. Type TEXT, a space, then a message from one to ten letters long. If you have a longer message, break it up into a few short ones.

You can make Elmer stop moving anywhere in the list with the instruction STOP. He'll hold still until you press any key, then continue on. It's a good idea to make STOP the last instruction on a list, so you can see what you and Elmer have created.

CHANGING THE LIST

You can alter Elmer's instructions in various ways after you've written them. As I mentioned earlier, NEW erases the whole list. To erase only one line, specify the line number, then type D for delete. The instruction at that line number will disappear, and all the ones below it will move up.

If you need to change an instruction instead of erasing it, type the new version after designating the line number. To add a line between two others, type the number of the second and an I for insert. That line will clear,

and its instruction and all those following will move down. Then, by accessing that line number again, you can fill in the blank. By the way, if you should leave blank lines in the list, Elmer won't mind. He'll just ignore them and plod along.

At any time you're on the second menu screen, you can change the line number you're working at by typing B, for Back to Line #. This recalls the first menu screen, where you can choose another line number.

You know, Elmer may not be very smart, but he has endless patience. He never tires of reading your list and walking around the screen following directions. He tells you when he doesn't understand, and gives you as many tries as you need to get it right.

Nowadays, when I'm done using our Commodore, I load in Elmer the Turtle and leave it on. More often than not, some visitor passes by and starts to play. Bingo!—another programmer is born. ☐

RUN Basic

By Robert Rockefeller

RUN It Right

C-64; Basic 4.5

RUN Basic, an extended Basic for the C-64, adds 30 new commands to Basic 4.5 (published in *RUN*'s June, July and August 1985 issues). RUN Basic includes commands for graphics, structured programming and named subroutines with local variables, among other features. It is aimed at fairly competent programmers, and even they will need to closely examine the demo programs to learn how to use these new powerful commands effectively.

To load RUN Basic 1.0 from disk, just execute:

```
LOAD "0:RUN BASIC 1.0",8,1
```

Once the program loads, enter SYS64738 and press the return key to start it.

TURTLE GRAPHICS

Most of RUN Basic's commands are for turtle graphics, so we'll start with a description of these. In most forms of turtle

graphics, the cursor is a crude representation of a turtle, to be imagined as holding a pen. You move the turtle with commands such as

```
AHEAD 30: TURNT0 HEADING + 1:  
BACK 59
```

When the turtle pen is down, you may draw or erase lines on the screen; when it's up, you can move the turtle without having it draw anything.

RUN Basic doesn't use a turtle because it slows the drawing process. For the sake of consistency, however, I refer throughout this article to the current position of the drawing pen on the hi-res screen as a turtle.

When plotting turtle graphics, RUN Basic uses a standard Cartesian coordinate system with the origin (0,0) at the lower left of the screen. The screen is 320 pixels wide and 200 pixels high.

Many of RUN Basic's turtle graphics commands require you to supply a color in the parameter string. The colors are specified as follows:

0—black	8—orange
1—white	9—brown
2—red	10—light red
3—cyan	11—dark gray
4—purple	12—medium gray
5—green	13—light green
6—blue	14—light blue
7—yellow	15—light gray

TURTLE GRAPHICS COMMANDS

HIRES. Command format: HIRES <screen-color (0-15), plot-type-color #1 (0-15)>.

HIRES initializes a high-resolution bit-map graphics screen, but it does not clear the screen. This permits the Hi-res mode to change the current colors while drawing. It also permits all 16 colors to be displayed simultaneously on the high-resolution screen. Each block of the hi-res screen is eight pixels wide and eight pixels high and can have a unique screen and plot color combination. Sample RUN Basic line:

```
10 HIRES 1,0 : REM WHITE HI-RES
    GRAPHICS SCREEN WITH BLACK
    PIXELS
```

MEDRES. Command format: MED-RES <screen-color (0-15), plot-type-color #1 (0-15), plot-type-color #2 (0-15), plot-type-color #3 (0-15)>.

MEDRES is similar to the HIRES command except that it initializes a multicolor graphics mode. In Med-res mode, each

four-pixel-wide by eight-pixel-high block of the med-res screen can simultaneously display three different colors, plus the screen color. By using MEDRES to select new drawing colors, all 16 colors can appear on the same med-res screen. Sample RUN Basic line:

```
10 MEDRES 0,7,3,4: REM YELLOW,
    CYAN & PURPLE DRAWING
    COLORS ON A BLACK SCREEN
```

TEXT. Command format: TEXT.

The TEXT command reverts the screen from Hi-res or Med-res mode to the text screen. Sample RUN Basic line:

```
10 TEXT: REM TO LOW-RES SCREEN
```

GCLR. Command format: GCLR.

GCLR clears the graphics screen. This command should be executed at the beginning of every RUN Basic turtle graphics program, to clear the graphics screens. Sample RUN Basic line:

```
10 GCLR: REM CLEARS HIRES &
    MEDRES SCREENS
```

PEN. Command format: PEN <plot-type (0-3)>.

PEN determines whether RUN Basic's pen will draw or erase when lowered. Following is a description of each plot type.

0—Erases with screen color in Hi-res or Med-res mode.

1—Plots in Hi-res mode with the plot-type color selected in the HIRES command. In the Med-res

mode, drawing will be done with plot-type color #1.

2—Plots in Med-res mode with Med-res color #2.

3—Plots in Med-res mode with Med-res color #3.

Sample RUN Basic lines:

```
10 PEN 0: REM LINE ERASES AS IT  
   PLOTS
```

```
10 PEN 3: REM LINE DRAWS AS IT  
   PLOTS
```

PENUP. Command format: PENUP.

PENUP is used to stop drawing. After PENUP is executed, drawing commands such as AHEAD, BACK and MOVXY will not draw. RUN Basic's turtle moves to its new location without drawing a line. Sample RUN Basic line:

```
10 PENUP: REM STOP PLOTTING  
   LINE HERE
```

PENDOWN. Command format: PENDOWN.

PENDOWN is the Default mode and allows lines to be drawn when RUN Basic's turtle is moved. Sample RUN Basic line:

```
10 PENDOWN : REM RESUME OR  
   BEGIN PLOTTING HERE
```

AHEAD. Command format: AHEAD <number of units>.

AHEAD moves the turtle the specified number of pixels ahead. Sample RUN Basic line:

```
10 AHEAD 30.5 : REM MOVE TUR-  
   TLE 30.5 PIXELS FORWARD
```

BACK. Command format: BACK <number of units>.

BACK moves the turtle the specified number of pixels backwards. Sample RUN Basic line:

```
10 BACK 162: REM MOVE TURTLE  
   BACK 162 PIXELS
```

MOVXY. Command format: MOVXY <x-coordinate, y-coordinate>.

MOVXY moves the turtle to the position specified by the x,y coordinates. If the pen is down during the execution of MOVXY, a line will be drawn. Sample RUN Basic line:

```
10 MOVXY 10,39: REM MOVE TUR-  
   TLE TO THE INTERSECTION OF X  
   (10) AND Y (39)
```

MOVX. Command format: MOVX <x-coordinate>.

This command moves the turtle to the specified x-coordinate. Sample RUN Basic line:

```
10 MOVX 100: REM MOVE THE TUR-  
   TLE TO THE X-COORDINATE (100)
```

MOVY. Command format: MOVY <y-coordinate>.

This command moves the turtle to the specified y-coordinate. Sample RUN Basic line:

```
10 MOVY 199 : REM MOVE THE  
   TURTLE TO THE Y-COORDINATE  
   (199)
```

TURNTO. Command format: TURNTO <angle>.

TURNTO points the turtle in the specified direction. The angle

must be in radians, as used in Basic 2.0. A radian is a unit of measure for angles or arcs. To convert degrees to radians, use the following equation:

Radian measure of an angle =
number of degrees $\times \pi/180$

An angle of 0 (radians or degrees) points the turtle to the right, on a heading parallel to the x-axis. An angle of $\pi/2$ radians, or 90 degrees, points it straight up, parallel to the y-axis. Sample RUN Basic line:

```
10 TURNT0 .4 : REM POINTS THE
    TURTLE UP AT AN ANGLE OF 23
    DEGREES MEASURED
    COUNTERCLOCKWISE FROM THE
    0 DEGREE DIRECTION.
```

LEFT. Command format: LEFT
<angle>.

LEFT turns the turtle left from its current direction by the specified angle (always in radians). Sample RUN Basic line:

```
10 LEFT .3 : REM TURNS THE TUR-
    TLE 17 DEGREES TO THE LEFT
```

RIGHT. Command format:
RIGHT <angle>.

This command turns the imaginary turtle right by the specified angle. Sample RUN Basic line:

```
10 RIGHT  $\pi/4$ : REM TURNS THE
    TURTLE 45 DEGREES TO THE
    RIGHT
```

PUTCHAR. Command format:
PUTCHAR <screen-code-value
(0-255)>.

The PUTCHAR command draws the specified character at the current turtle position. The character is specified by its screen-code value, not its ASCII value. Check your user's guide for a table of screen codes. Sample RUN Basic line:

```
10 PUTCHAR 1: REM PRINT AN "A"
    AT CURRENT POSITION
```

HOME. Command format:
HOME.

HOME moves the turtle to the center of the screen. Sample RUN Basic line:

```
10 HOME : REM MOVE TO CENTER
    OF SCREEN
```

PLOT. Command format: PLOT
<plot-type (0-3), x-coordinate
(0-319), y-coordinate (0-199)>.

The PLOT command plots one pixel at the specified coordinates using the specified plot-type (see definitions of plot-types under the PEN command, above). The location of RUN Basic's imaginary turtle has no effect on this command. Sample RUN Basic line:

```
10 PLOT 1,100,38 : REM PLOT A VISI-
    BLE PIXEL AT X (100), Y (38)
```

XPOS. Command format: XPOS.

XPOS returns the current x-coordinate of the turtle's position. Sample alternative RUN Basic lines:

```
10 A=XPOS : PRINT A: REM PRINTS
    CURRENT X POSITION OF TURTLE
10 PRINT XPOS : REM PRINTS CUR-
    RENT X POSITION OF TURTLE
```

YPOS. Command format: YPOS.

This command returns the current y-coordinate of the turtle's position. Sample alternative RUN Basic lines:

- ```
10 A=YPOS : PRINT A: REM PRINTS
 CURRENT Y POSITION OF TURTLE
10 PRINT YPOS : REM PRINTS CUR-
 RENT Y POSITION OF TURTLE
```

**HEADING.** Command format: HEADING.

HEADING is used to determine the angle of the direction in which the turtle is currently pointing. Sample RUN Basic lines:

- ```
10 A=HEADING : PRINT A : REM  
    PRINTS CURRENT ANGLE TURTLE  
    IS POINTING  
10 PRINT HEADING : REM PRINTS  
    CURRENT ANGLE TURTLE IS  
    POINTING  
10 TURNT0 HEADING+.1 : REM  
    TURN TO CURRENT HEADING  
    PLUS .1 RADIANT
```

COORDS. Command format: COORDS (x-coordinate, y-coordinate).

This command returns the direction from the turtle's position to the specified coordinates. Sample RUN Basic lines:

- ```
10 A=COORDS(0,0) : PRINT A : REM
 PRINTS THE DIRECTION OF THE
 POINT (0,0) FROM THE TURTLE'S
 POSITION
10 PRINT COORDS(31,29) : REM
 PRINTS THE DIRECTION OF POINT
 (31,29) FROM THE TURTLE'S
 POSITION
10 TURNT0 COORDS(150,36): REM
 TURNS TURTLE TOWARDS (150,36)
```

**RPIX.** Command format: RPIX (x-coordinate, y-coordinate).

RPIX is used to return the plot-type used to draw a pixel. The plot-type returned in Med-res mode will vary from 0 to 3. In Hi-res mode, the plot-type returned will be either 0 or 1. Sample RUN Basic lines:

- ```
10 A=RPIX(300,150): PRINT A : REM  
    PRINTS THE PLOT-TYPE USED AT  
    300,150  
10 ON RPIX(QT,QS) GOSUB 100,200 :  
    REM GOSUB WHEN THE ARGU-  
    MENT IS SATISFIED
```

SPRITE COMMANDS

DATCOLL. Command format: DATCOLL.

DATCOLL returns the contents of the sprite-background collision register. If a sprite collides with the background, its corresponding bit is set. Bit 0 corresponds to sprite 1, bit 1 corresponds to sprite 2, etc. Sample RUN Basic lines:

- ```
10 IF DATCOLL AND 128 THEN GO-
 SUB 1000: REM HANDLE SPRITE 8
 COLLISION
10 A=DATCOLL : PRINT A : REM
 PRINTS SPRITE/BACKGROUND COL-
 LISION OCCURRED
10 PRINT DATCOLL : REM PRINTS
 SPRITE/BACKGROUND COLLISION
 OCCURRED
```

**MCOLL.** Command format: MCOLL.

MCOLL returns the contents of the sprite-sprite collision register. When two sprites collide, their cor-

responding bits are set. The sprites are then mapped to the register bits in a slight variation of the method explained under DAT-COLL. The difference is that the collision of two sprites sets two bits instead of one. Sample RUN Basic lines:

```
10 IF MCOLL = 4 + 2 THEN GOTO 6000
 : REM SPRITES 3 AND 2 HAVE
 : COLLIDED
10 IF MCOLL = 8 + 16 + 32 THEN GOTO
 4000 : REM SPRITES 4, 5 AND 6
 : HAVE COLLIDED
10 PRINT MCOLL : REM PRINTS IF
 : SPRITE/SPRITE COLLISION OCCURS
10 A=MCOLL : REM IDENTIFIES
 : SPRITE/SPRITE COLLISION
 : OCCURRED
```

## STRUCTURED PROGRAMMING

**SUB.** This is used to declare a named subroutine that can be executed with the CALLSUB command. The subroutine's name must follow the SUB command and may be up to 31 characters long, though only the first four letters are significant.

Unlike variables, Basic commands can be embedded within the name. SUB PRINT-IT(A\$) is an acceptable subroutine title. Punctuation and spaces can be used within the name, but graphics characters cannot.

Parentheses that enclose an optional list of variables must follow the name. The optional list of vari-

ables contains any values to be passed to the subroutine. This passing of values is similar to the Basic 2.0 DEF FN command, except that up to 35 values may be passed to a subroutine; only one value may be passed to a function.

When the SUB command is executed, it scans ahead in the program, looking for the SUBEND command, which will be described next. If SUBEND is not found, a Syntax error occurs.

If SUBEND is found, the subroutine is initialized, and program execution is resumed at the line following SUBEND. The code between SUB and SUBEND is executed only when the subroutine is called with the CALLSUB command.

The variables enclosed by parentheses following the subroutine name are local variables, meaning that they are referenced only when the subroutine is executed. After the subroutine finishes executing, all local variables created by the subroutine are no longer accessible.

This allows you to use the same variable name twice—once within the subroutine and once outside the subroutine.

**SUBEND.** SUBEND is used to mark the end of a subroutine and is equivalent to the RETURN command used in Basic 2.0 subrou-



tines. The guidelines for SUBEND are simple.

First, every SUB must have its corresponding SUBEND. Next, SUBEND must be the first command on its line. Finally, if the SUBEND command is preceded by colons or any other command, SUB will be unable to locate it, and a syntax error will occur.

The following sample program demonstrates the use of the SUB, CALLSUB and SUBEND commands:

```
10 SUB TEST (X,Y,A$)
20 PRINT "LINE 20
 X="X;"Y="Y;"A$="";A$
30 SUBEND
40 '
50 X=5: Y=5: A$="GLOBAL"
60 PRINT"LINE 60
 X="X;"Y="Y;"A$="";A$
70 CALLSUB TEST(1,2,"LOCAL")
80 PRINT "LINE 80
 X="X;"Y="Y;"A$="";A$
```

In line 50, three global (normal) variables, X,Y,A\$, have been assigned values. Three local variables with the same names are created by CALLSUB when it calls the subroutine TEST. The local variables are different variables and have different values from the global (normal) variables, but they share the same names!

How does this program execute? The SUB command is executed first, and it scans ahead looking for SUBEND, which it finds on line 30. It then initializes the

subroutine TEST. This permits execution to resume on line 40. The code from lines 40 to 80 is executed normally.

The output appears as follows:

```
LINE 60 X=5 Y=5 A$=GLOBAL
LINE 20 X=1 Y=2 A$=LOCAL
LINE 80 X=5 Y=5 A$=GLOBAL
```

This program demonstrates how global variables retain their assigned values, despite the fact that the local variables were assigned different values within the subroutine.

**EXIT.** This is used to terminate subroutine execution prematurely. It is the equivalent of the RETURN command. Like SUBEND, EXIT works only with named subroutines. See the subroutine CURVE in the demo program for an example.

**LOCAL.** This is used to create local variables in addition to those CALLSUB has created and passed values to. LOCAL should only be used within a named subroutine. Sample RUN Basic line:

```
LOCAL A,B,FF%,QQ$
```

## TECH TALK

RUN Basic works by using the C-64's RAM in a variety of ways. The RAM under the Basic ROM is used for storage of variables. A graphics bit map uses the RAM under the Kernal ROM. RUN Basic's turtle graphics com-

mands for color memory use the RAM at addresses 49152 to 50176. Free RAM exists from locations 50177 to 53247 for other uses such as sprite storage.

To see a demonstration of RUN Basic's turtle graphics and

structured programming commands, run demos 1 through 6. You'll be impressed with the speed of RUN Basic's turtle graphics, compared to other turtle graphics programs. 

# Programmers, Take Note!

By Bob Kodadek

## **RUN It Right**

*C-64*

While computing, how often do you have to spend time looking up Poke, Peek or SYS numbers because you forgot them? Or how often have you had an idea you wanted to write down quickly but were unable to find a pencil or paper handy? If you're like me, you end up making a lot of mental notes, then forgetting them.

With 64 Notepad, you have an electronic pad, pencil and eraser at your fingertips.

The program adds a text window to your screen. The window has editing capability and does not affect your present screen. You can access it at the touch of a key, open and use it even while another program is running. When you close the window, your program continues

without missing a byte, and your notes are safely stored in a memory that never forgets.

### **ABOUT THE PROGRAM**

After you run Notepad, press the CTRL-O key combination to open the window. When it first opens, the notepad will be filled with garbage; simultaneously press the shift and CLR/home keys to clear it. The notepad consists of 15 lines, each 38 spaces long. You may change your text color within the window by pressing CTRL-P.

The home, return, cursor and delete keys function as you would expect. The insert key, however, is disabled.

To close the window and return to the previous display, press CTRL-C. Simultaneously press the run/stop and restore keys to disable the utility. To restart, simply enter SYS 51072.

This program can be used with



Robert Rockefeller's RUN Basic. Also, if you want to print out your notes, you may do so using RUN's previously published screen dump utility (see "Print Your Screen," December 1984), which runs concurrently with

Notepad, as does RUN's 64 Perfect Typist checksum utility, the DOS wedge or any program that doesn't conflict with the memory area from 51072 to 51852.

Now get busy and take some notes! 

# Sign Maker

By Ken Amberg

## **RUN** It Right

*C-64; C-128 (in C-64 mode); VIC-20  
Printer*

I own a printer that has an Enlarge mode for printing larger characters. The Enlarge mode is nice, but I wanted letters big enough to be visible across a room.

Sign Maker prints characters in two ways. One format produces the largest letters and prints them sideways down the paper, banner style, up to 40 characters high. The other format prints smaller characters on a standard 8½ by 11-inch sheet of paper.

The menu appears on the screen after you load and run

the program. Press the F1 key to use the banner maker. Press the F3 key to use the single-sheet sign maker. The F5 key prints a form-feed character, and the F7 key ends the program and closes the print file.

To make a sign, simply answer the different questions the program asks.

I've found plenty of uses for this little printer utility program. It's great for birthday messages and greetings, and my family always sees the notes I leave on the refrigerator. Experiment with this program; it could breathe new life into your printer, which is useful for more than just making program listings and reports. ☐

# Instant Data Statements

By Thomas H. Simmonds, Jr. and Jim Borden

## **RUN** It Right

*C-64; C-128*

Datamaker 64 and 128 enable you to incorporate machine-language routines into your Basic programs. They peek anywhere in random access memory and transcribe the hexadecimal machine language there into lines of Basic Data statements. The transcriber then deletes itself, leaving the Basic Data lines for you to add to or merge with an existing program. This is an ideal way to add custom characters or sprites to a program after they've been generated by an editor program.

### **THE C-64 VERSION**

Use a "filename",8,1 format to load RAM with the machine language you wish to transcribe into Data statements. Now load Datamaker 64, run it, and type in the information requested. Be sure the starting Basic line number is greater than the highest

line number in the transcriber. You may want to choose line numbers and a line-number increment that will be compatible with the program to which you plan to add the Data statements.

Then enter the beginning and ending RAM locations to be transcribed. The transcriber will cycle through, writing Data statements and then line numbers to the screen until it has processed the last memory location you designated. When it's done, save the statements to disk, clear the computer and load the program you are adding the statements to. Then, in Direct mode, type the following as one line and press the return key:

```
L=256*PEEK(46)+PEEK(45)-2:LH=
INT(L/256):LL=L-256*LH:POKE43,LL:
POKE44,LH:CLR
```

Next, load the Data statements and, in Direct mode again, enter the following line and press return:

```
POKE43,1:POKE44,8
```



MEMORY TO DATA ST.

ENTER FIRST DATA LINE NUMBER

? 500

ENTER LINE NUMBER INTERVAL

? 10

ENTER FIRST LOCATION IN MEMORY TO BE  
PUT IN DATA ST.

? 49152

Now list the program. It should include the Data statements. Note that these two last steps can be used to merge any Basic programs.

### THE C-128 VERSION

Datamaker 128 takes advantage of some Basic 7.0 keywords, but also allows data to be read from any bank of memory. In the 128 version, the F1 key replaces the dynamic keyboard.

To add the Data statements to the main program, first load the main program, then, in Direct mode, type:

```
GRAPHIC CLR:T=65278-FRE(0):
POKE46,T/256:POKE45,
T-PEEK(46)*256
```

and press the return key. Next, load the Data statements and

then type, again in Direct mode:

```
POKE45,1:POKE46,28
```

and press return. Now you can list the combined program. As with the C-64 version, you can use this procedure to combine any Basic programs.

### VARIABLE STORAGE

A note on how variables are stored is in order here. You'll notice that the C-64 version of the transcriber has to poke all the variables and later peek them, but the 128 version does not. The 64 stores numeric variables at the end of a Basic program, where they may be overwritten if a line is changed. That's why Basic does a CLR in the 64 any time you revise a line.

With the transcriber, if the variables weren't poked into memory, they'd all contain a value of zero after the program read in the first line of data. The 128 uses a separate bank of RAM to store its variables, and Basic doesn't affect it when changing, adding or deleting lines.

Users of the C-128 should keep this in mind. It's usually possible to change a line and then go right on executing the program. If, for example, you get a Syntax error in line 500, you can list the line and correct the error, then continue the program by typing GOTO500 and pressing the return key. This is a

way to avoid long setup delays while debugging programs, a trick you'll appreciate more and more as your programming skill improves.

Using this transcriber utility to make Data statements is an easy way to build a library of subroutines, each starting at a different line number, that can later be incorporated into a main program. By keeping a list of the starting line numbers, you can then include GOSUBs to any of them when you're writing a main program. After you append the subroutine and renumber, Basic 7.0 will handle all the GOTO and GOSUB addresses for you. 

# 64 Personal Ledger

By Paul Beddows

## **RUN** It Right

*C-64; disk drive; printer optional*

A program that keeps track of savings accounts or transactions involved with running a small business out of the home can be useful. And if you're the treasurer of a club or organization, it can take a lot of the drudgery out of the bookkeeping.

Balance Sheet is such a program, and it lets you keep several years' records on a single disk; so if the IRS ever decides you deserve an audit, you should be in good shape.

Balance Sheet should be stored as the first and only program on your disk. It will create sequential files on the same disk, in which your data will be stored.

When you load and run the program, you'll be prompted for a filename—a bank account number or any other filename of ten characters or less. Filenames of more than ten characters will be truncated.

Press the return key and an-

other prompt will appear. If you're establishing a file for the first time, enter \* and press the return key again. The main menu will appear. If the file already exists, press the return key without first entering \*, and the file should load.

File length, by the way, has been set to 95 entries to shorten loading and saving times. You may change this value by altering the value of X in line 50. Since the program can carry over balances from one file to another, I have found 95 entries to be more than sufficient.

## **MENU OPTIONS**

*F4:Enter Information.* There are four fields of data entry: Date, Item, Debit and Credit. The date must be entered in numerical format and mustn't be more than six characters long. An attempt to enter a non-numerical format will default the date to a "--" symbol. Entries of more than six characters will be truncated.

Dates must be entered using the format "year/month/day" or



"month/day" in order for the chronological sorting feature to work. Days and months with one-digit numerical equivalents must be preceded by a 0. For example, April 3, 1986 should be entered as 860403 or 04/03.

Starting with the second entry, the previous entry's date is automatically displayed. This makes it easier to enter multiple entries for a single day. Either press the return key to confirm the date or type a new date over it. You may eliminate this feature by deleting D\$(N-1) from line 520.

Following the date, you may enter into the Item field a brief description (28 characters maximum) of the entry or just leave it blank by pressing the return key. Don't use punctuation marks that are invalid in Input statements, such as commas or colons.

Next, fill in either the Debit or Credit fields, or both. Both default to 0 if nothing is entered. Don't use dollar signs, commas or negative numbers; the program keeps track of these. Amounts in excess of \$100,000 will disrupt the Balance Sheet screen display, although the program will still operate on them.

Once all four data fields have been filled, the balance is automatically updated and displayed. At this point, you have the option of adding another entry, returning

to the menu, reentering the last entry or erasing the last entry.

*F1/-:Scan Forward/Reverse.* Once a file contains entries, you may scan them forwards or backwards by holding down the F1 or the left-arrow key. When an individual entry is displayed, you may modify or erase it by using one of the options displayed at the bottom of the screen. Each time you erase or change an entry, the balance is automatically recalculated through the subsequent entries.

*F5/F6:Balance Sheet.* This function displays the entries in column format, with the exception of the Item field, which couldn't be included because of the C-64's 40-column limitation. Pressing the F6 key takes you directly to the last page. You may access individual entries from this screen by pressing the R key, followed by the entry number.

You may also go directly into the Scan, Enter Information or Column Totals modes from the balance sheet display without first returning to the main menu.

*Column Totals.* Pressing the equals sign will add up the total debits and credits in a file.

*S:Save.* Following any additions, changes, deletions or chronological sorts, press the S key to resave the file to disk.

## Balance Sheet

F1 Forward Scan   ← Reverse Scan  
F4 Enter Information  
F5 Balance Sheet  
F6 Last Page

= Total Debits/Credits

\* Select New File  
↑ Rename File  
% Re-Establish File  
  
S Save  
C Chronological Sort  
P Print  
+ Exit

-----  
File: Budget '86      Entries: 0  
Balance: 0            Last Date:

*↑:Rename File.* Press the ↑ key to alter the name of a file. You may also use this option to create a backup file under a different filename. To accomplish this, rename the file, but do not execute it to disk when prompted. Press the S key when the main menu reappears.

*\*:Select New File.* Pressing \* clears the file in memory and returns you to the introductory screen.

*C:Chronological Sort.* After updating a file, you may sort the entries chronologically. This may

take a while if a large number of entries is involved. The screen border will flash during the sorting process. Remember to re-save to disk to make the change permanent.

*P:Print.* You may print out a file by pressing the P key. This option works with most printers. However, it won't function properly on printers lacking the tabulator function. The file can be printed partially or in its entirety.

A paging option is also provided for printers that recognize the "advance-to-top-of-form"



command (CHR\$(12)). The number of entries per page has been set at 56 in line 2880. This allows for a fair degree of error in positioning the paper. If you wish, you may increase this by four or five entries. Page 1 automatically has five fewer entries to allow space for the headings.

To accommodate non-Commodore printers and interfaces, you may have to make some alterations to the print routine. Commodore printers perform an automatic linefeed after printing a line, and the Print routine expects this. If your printer is overprinting lines, you'll have to add a linefeed after line 2910. For example:

```
2915 PRINT#4,CHR$(10)
```

If the paging option won't work on your printer, try replacing line 2920 with:

```
2920 IF SP = 56 THEN FOR ZZ = 1 TO
 10:PRINT#4,CHR$(10):
 NEXT:SP = 0
```

With this substitution, your printer should do ten linefeeds at the bottom of each page. If it's doing 20, then your interface is adding an additional linefeed whenever it receives a linefeed command from the program. Solve this by changing "zz = 1to10" to "zz = 1to5".

*%:Reestablish File.* You use this function when a file is full or if you wish to eliminate the older

entries. (The file must contain at least five entries for this function to work.) This is the most confusing and dangerous function of the program. I suggest you create a dummy file and try it out before using it with a real file.

This operation establishes a new file under the same filename as the old one, with the current balance carried forward along with the number of entries you select. If you wish to preserve the old file, you'll be prompted to enter a new filename for it, as the old name is being reused for the new file. (I usually use the old name, preceded by a "-".) Regardless of whether you choose to preserve or erase the old file, you'll be given the option to print a hard copy of it.

After the program loops through the Print routine, you'll be prompted to select the number of entries you wish carried forward into the new file. (Should you change your mind at this point, enter a number outside the given range, and you'll default back to the main menu.) The old file will be erased and replaced by the new one, with the balance forward showing as the first entry. The date on the balance forward will appear as "\*\*\*\*\*" to ensure it always remains as the initial entry in the file, following any subsequent chronological sorts. 



# High-Resolution Revolution

By Henrik Markarian

## RUN It Right

C64

High-resolution color graphics is one of the C-64's main features, yet C-64 Basic doesn't provide any commands to let you easily take advantage of it. For example, there is no command to clear the high-resolution screen. Instead, you must poke 8000 memory locations with zeros; in Basic, this takes about 30 seconds. Plotting a single point requires a lot of computation and time.

The 64/Grafix program augments 64 Basic with eight commands that make it easy for you to draw high-resolution pictures. The program is written entirely in machine language, although it is in the form of a Basic loader.

### 64/GRAFIX COMMANDS

You must precede each command with an exclamation point to let the 64 Basic interpreter

know that it is a special command. If the commands follow an *If...Then* statement, you must precede them with a colon, too, or an error will occur.

**ISC,n**—This command turns the high-resolution screen on or off, depending on the value of *n*. If *n*=0, then the high-resolution screen is turned off, and the regular text screen displayed. If *n*=1, the high-resolution screen is turned on but not cleared. Setting *n*=2 will turn on the high-resolution screen and clear it.

**IPN,n**—This command turns the pen on or off, depending on the value of *n*. If *n*=0, then the pen will be off, and points (pixels) will be erased; if *n*=1, the pen will be on, and points (pixels) will be plotted.

**ICO,n**—This command colors in a background and foreground color on the entire graphics screen. The parameter *n* must be within the range of 0-255, and

it is determined by the following formula:

$$n = (\text{background color}) + (\text{foreground color}) * 16$$

For example, if you wanted the background red and the foreground blue,  $n$  would have to equal  $2 + 6 * 16$ , or 98.

**IBL,b,n**—The Block command lets you choose the background and foreground color for any  $8 \times 8$  block on the graphics screen. The screen is divided into 1000 blocks, which are numbered from 0 to 999, starting at the top left-hand corner and moving right. The first parameter in this command is the block number ( $b$ ), and it is determined in the following fashion:

$$b = (\text{column}) + (\text{row}) * 40$$

Of course, on the C-64 there are 25 rows and 40 columns. The second parameter for this command ( $n$ ) sets the background and foreground colors for the specified block, using the same form as the previous command.

**!PL,x,y**—This command plots a point at the coordinates  $x$  and  $y$ , or erases a point at the same coordinates. It all depends whether the pen is on or off (see the !PN,n command). The parameters  $x$  and  $y$  are offset from the top-left corner of the screen and have a range of 0-319 for  $x$  and 0-199 for  $y$ .

**IDR,x<sub>1</sub>,y<sub>1</sub>,x<sub>2</sub>,y<sub>2</sub>**—This command draws a line from point  $x_1, y_1$  to the point  $x_2, y_2$ . As before, the line is plotted or erased depending on whether the pen is on or off.

**!CR,x,y,r**—This command draws an oval, given the center coordinates  $x, y$  and the radius  $r$ . The radius must be within the range of 0-255, although, as you'll see in most cases, the largest possible radius will be around 110-130. To make this command more versatile, you may multiply the vertical distance by a fraction, allowing you to draw anything from a wide short oval to a tall narrow one. The following two memory locations are used to input the numerator and denominator for the fraction ( $N/D$ ):

POKE 50243,N

POKE 50251,D

The program at this point assigns  $N/D = 9/11$ . I arrived at this fraction by measuring a few pixels parallel to the  $x$ -axis and the same number of pixels parallel to the  $y$ -axis. From these measurements, I calculated that the length of nine pixels in the horizontal direction is equal to the length of 11 pixels vertically. Thus, multiplying a given vertical distance by  $9/11$  will make the ovals appear more like true circles than ellipses.

**!EX**—This command exits 64/Grafix from the command-pro-

cessing loop. You can reactivate the program with SYS 49152.

### **FOR YOUR INFORMATION**

The 64/Grafix program resides in the 4K of RAM from \$C000 to \$C4D0. The area from \$C800 to \$CBFF is used as the color screen, and the RAM under the operating system ROM from \$E000 to \$FFFF is used for the graphics screen. 64/Grafix takes no memory space away from Basic; the area from \$0800 to \$9FFF remains completely free for Basic programs.

I've included two demo programs to help you further understand and use the 64/Grafix commands. The first demo program draws several figures on the screen and then waits for a keypress. If you want to exit the program, press F1; otherwise, press any other key to continue.

The second demo is a joystick-controlled (port #2) drawing program. F1 clears the screen and sets the background and foreground colors; F3 clears the screen; F5 exits the program. 



# As the Word Turns

By Gerald Caron

## **RUN** It Right

*C64; VIC-20; printer*

Find the Word 3.0 is a revision of my Find the Word program that appeared in the October 1984 issue of *RUN*. After it appeared, I received many letters from people who had been looking for such a program, and many fellow teachers wrote to thank me for my efforts. Some writers offered suggestions; others wished the program had additional functions.

One of the most requested functions was the ability to save and load puzzles to avoid having to type all the words in again. I have provided this by using sequential files. With this feature, the puzzle also prints out immediately after loading, and there's no waiting for set-up or placements.

I have added disk error trapping, as well, to help prevent crashes, and now include a .WS in the name of the puzzle file so you can spot it more easily in

the directory. You don't have to add the .WS when you load a puzzle because the program does it automatically.

The routines for these procedures are located after line 1000 in the program listing. They store the grid and the word direction and puzzle size options that I'll discuss shortly.

Another feature I've added is an option to have the same words in a different grid. If you ask for a new grid, all the arrays except the word array have to be reinitialized. This is taken care of starting at line 1200. Now you can use the same words over and over, but get entirely different puzzles.

A word-direction option was something I wanted to add after a teacher at my school told me the reversed and slanted words were too difficult for younger children. Then I received a letter from a reader who had added that option plus one for puzzle size. Realizing that a small puzzle with words going only across or up and down could be done by almost anyone,

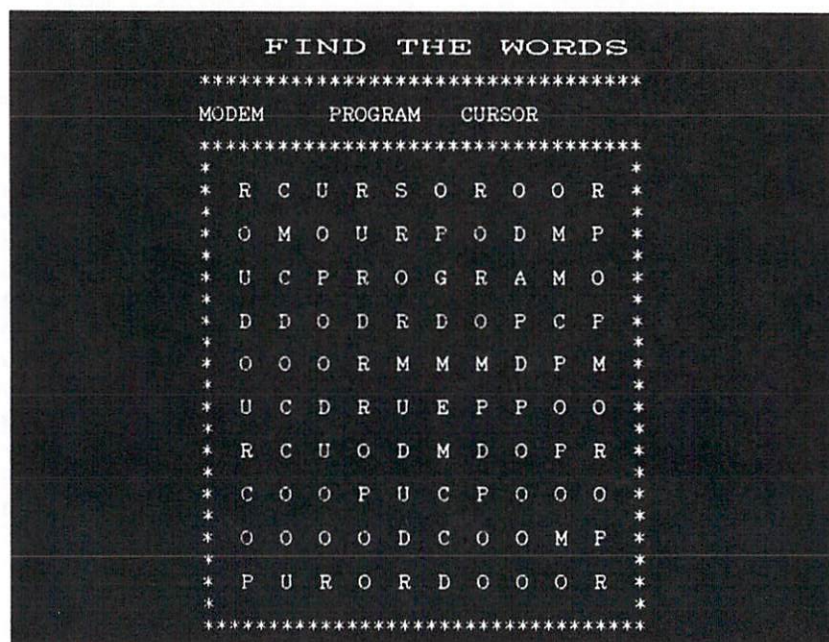


Figure 1. An example of a 10 by 10 puzzle.

I added an option for smaller puzzles. The grids now can range from 10 by 10 to 20 by 20, with the number of words adjusted for the puzzle size.

You can also make puzzles with numbers or graphics characters. Figure 1 is an example—a 10-by-10 puzzle composed of letters, numbers and graphics characters.

Many people wrote saying they were using the puzzles for everything from church bulletins to TV fact magazines. At my school, we always put one in our monthly school newspaper. If you are using the puzzles in an interesting way, I'd like to hear about it. I'd also welcome any ideas you have for further enhancements to the program. [E]

# Making a Pointer

By John M. Campbell

## **RUN** It Right

*C-64; joystick in port 2*

Demo Pointer, written in Basic, places on the screen a pointer in the form of an arrow. You control the pointer's movement with a joystick plugged into port 2.

You can use Demo Pointer with other Basic programs to point to locations in text or charts on the screen. This is especially helpful at club meetings or in the classroom.

Before running Demo Pointer, plug your joystick into port 2. Pushing the fire-button once turns the pointer off, and pushing it again turns it back on. If you move the pointer off one side of the screen, it will reappear immediately on the other.

Once you run the program, you'll be prompted to select the size and color of the pointer. The program will then poke into memory a short machine language program that controls the pointer's movement independently of what

ever else you've loaded into the computer's memory.

The C-64 allows you to define up to seven sprites for display on the screen. The pointer arrow is just such a sprite. The pattern of bits that compose it are contained in the Data statements in lines 300-360. You may alter the form of the sprite from an arrow to a hand or a turtle simply by changing the numbers in those lines. *The Commodore 64 Programmer's Reference Guide* describes in detail how to define your own sprite.

## **PROGRAM LIMITATIONS**

Demo Pointer was written to be as flexible as possible, but it has a few limitations.

It will run with just about any Basic program except those containing sprites or using a joystick in port 2. Demo Pointer uses locations 679-767 to hold the sprite and perform other house-keeping chores, so it can't co-exist with other programs that use those addresses.



The actual machine language program is 175 bytes long and is loaded at address 49200. If you want to run Demo Pointer with another program that uses that section of memory, you may load the Demo Pointer machine language routine into a different part of memory simply by changing the address in line 370.

You can't move the arrow while either the disk drive or Datassette is active, as the computer is too busy to accept the clock interrupts. Moving the pointer while typing can cause entry errors; this is because the joystick and keyboard share a register in the complex interface adapter (CIA). 

# Double Vision

By Larry Cotton

## **RUN** It Right

*C-64; C-128 (in C-64 mode)*

Mirror, Mirror is an excellent first program to type into your Commodore 64. When you draw a pattern with the joystick, a mirror image is drawn simultaneously. This, complemented by randomness and color, yields beautiful, magical surprises, arising from the symmetry of the mirror image. All the while, a mellow "bipping" sound tracks the cursors, rising and falling in pitch as they move.

The liberal use of constants, which are defined early on, and a minimum number of `if...` Then statements ensure quick response to joystick movements, even though the program is written entirely in Basic.

A very short menu at the beginning of the program allows you to change patterns and colors. To erase drawing, press

the space bar, which brings you the simple menu again.

To use the program, turn up the volume on your TV or monitor. Plug the joystick into port 2 (next to the power cord) and run the program. Try pressing the joystick to the right. Both characters will move. You're controlling the *right-hand* one, unless they cross. If they do cross and continue past the right and left borders, the cursors will get out of sync vertically. You can bring them back into sync by backing up and retracing your pattern.

To avoid using a lot of `if...` Then statements, which slow down the program, there are only minimum top and bottom border checks. You may go to these borders, but if you keep pushing the joystick, the program will end. Press the space bar for a new start.

Now press the space bar and answer the questions differently. You will be truly amazed at the results! ☐

# Chaining Your Programs

By Michael Broussard

## **RUN** It Right

*C-64; VIC-20*

A friend of mine who is relatively new to programming is working on his most ambitious project to date—an all-text adventure game written in Basic for the Commodore 64. After a week or so, he discovered that the 38K of memory available for a single program would not be enough. "Now what do I do?" he asked me the other day. "I'm out of memory!"

"Well," I replied, "you'll have to use program chaining." I explained the fundamentals of chaining using Commodore Basic. It's quite simple, really. All you have to do is employ the Load command from inside a program. This reads in the new program specified from disk (or tape). When it's loaded, it wipes out the program currently in memory and then runs automat-

ically. As long as the "chained" program is shorter than the original, all the variables will be available to the chained program as well. I even wrote an example program to show him how it works:

```
10 PRINT"THIS IS THE MAIN
PROGRAM. NOW LET'S"
20 PRINT"CHAIN TO THE SUB
PROGRAM."
30 LOAD"SUBPROG",8
```

When you run this short program, it executes the Print statements shown and then loads a new program called SUBPROG from the disk. SUBPROG then runs automatically.

Armed with his new knowledge, my friend went back to programming, confident his memory problems were solved. But, within an hour he was back. "I've figured out how to break my game up into parts," he said, "but there's about 10K of code that's exactly the same for each



THIS IS AN EXAMPLE OF A MAIN  
PROGRAM.

PRESS ANY KEY TO LOAD IN  
THE OVERLAY.

THIS IS THE OVERLAY.

PRESS ANY KEY TO RETURN TO  
THE MAIN PROGRAM.

AND HERE WE ARE BACK IN THE  
MAIN PROGRAM.

part. Do I have to include that code as part of *every* chained program?"

Uh oh. Now the problem was getting harder. The fastest way out was to tell him that, yes, he had to include the 10K in every program. But it would take longer to load each part, not to mention the extra disk space each would occupy. Plus, for every change he might want to make to the "common" part of the code, he would have to change *every* program. Keeping all the parts in sync could turn into a time-consuming nightmare.

After a couple of hours of experimentation, I came up with a way to solve this dilemma. It's a pair of subroutines (one written

in Basic, one in machine language) that enables you to do program chaining while protecting a portion of the program already in memory. Throughout the rest of this article I'll refer to this utility as Overlay.

The only restrictions involved in using Overlay are that the area of memory to be protected (I'll refer to it as the common code) must be the first part of the program, and all the overlays must begin with the same line number.

My friend decided to keep all the common code in lines numbered less than 1000. He uses two overlays I'll call MAIN and SUB1, both of which start with line 1000. Overlay's utility subroutines are part of the common

code included in the lower-numbered lines. When the code in MAIN or SUB1 is needed, Overlay chains the appropriate program into memory starting at line 1000, leaving the beginning of the program (the common code) intact. First let's look at Overlay and how to use it—then I'll explain how it works.

### **OVERLAY SAMPLE PROGRAMS**

The best way to explain how to use Overlay is through an example.

Try loading and running OVERLAY.BOOT. Notice how the variable Z is initially set in the boot program (line 1050) and then is used to specify where the main program should begin executing before and after calling up an overlay.

After you see how these examples work, you'll be able to use overlays in your own programs with a minimum of hassle. Modify the boot program to suit your own applications. If you wish to start your overlays with another line number besides 1000, all you have to do is change line 1 of the boot program and the value assigned to the variable ZL in line 10.

### **HOW OVERLAY WORKS**

To understand how Overlay works, it's helpful to understand

how the 65536 (64K) bytes of memory in the C-64 are used when you're programming in Basic.

When you first turn on or reset the 64, you see the familiar power-on message that tells you there are 38911 Basic bytes free. What happened to the other 26625 bytes? They're there, but just not available for use by a Basic program. Look at the diagram in Figure 1.

If you think of the memory in the C-64 as a string of bytes numbered from 0 to 65535, your Basic program typically resides at the beginning of the area that starts at location 2048 and extends up to 40959. The "lower" part of RAM, from location 0 to 2047, is used by the Basic operating system to do housekeeping. This space includes memory for screen management, the cassette buffer, important Basic pointers, and so forth.

The Basic language and operating system program is stored on special ROM (Read Only Memory) chips, taking up 8K of memory in locations 40960 to 49151. Next comes a 4K chunk of memory from 49152 to 53247 that's free to be used for whatever you like—perhaps for machine language programs that must coexist with Basic.

Locations 53248 to 57343 are used for input/output and for

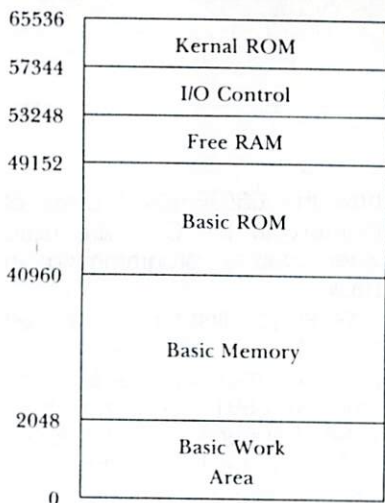


Figure 1. Typical memory allotment in the C-64.

sprite and sound control. Finally, at the very "end" of memory (57344 to 65535) is the Kernal ROM, a set of special machine language routines used by Basic for input/output, timer and memory management and other useful tasks.

This is, in a slightly simplified manner, how all the memory in the 64 typically is configured. Now let's turn our attention to the 38000 or so bytes reserved for your Basic program, which I'll refer to as Basic memory.

Basic memory contains not only your program, but also the values of all the variables and

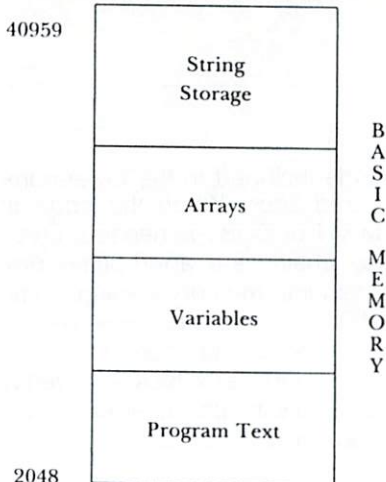


Figure 2. Basic memory areas in the C-64.

arrays the program needs as it runs. When you run a program, all variables are reset, and Basic creates space for each variable and array the first time it's referenced. Variables occupy memory just past the end of the program text, and arrays are just past the variables.

String variables are handled in a special way. Since strings can vary in length up to 255 characters, the string values go at the top of Basic memory, and only descriptive information about the variable (its name, length and where it's actually stored) is found with all the other variables.



The diagram in Figure 2 summarizes this information.

Special pointers in lower RAM keep track of the dividing lines between the various portions of Basic memory. RAM locations 43 and 44 contain the address where program text is to be loaded (the start-of-Basic pointer). As you can see in Figure 2, this is usually set to 2048 on the 64.

Overlay works by scanning the program until it finds line 1000. Then it stores the address of line 1000 in the start-of-Basic pointer and loads in the specified overlay program, starting at *that* address instead of the default location of 2048. This leaves the part of the program numbered less than 1000 (the common code) intact.

The overlay program begins executing at line 1000, because that's where Basic thinks the program starts. The first line of the overlay resets the start-of-Basic pointer to include lines 1-999 again, so those Pokes must be executed before you reference any of the common code.

When a program is loaded, Basic also sets another pointer at locations 45 and 46 to point two bytes past the program end. This is where the variables will be stored when the program is run initially. When you use Overlay, however, this may present a

problem. As you can see, if you load an overlay that's long enough to extend past where this pointer points, you'll destroy some of the program variables. To get around this, the boot program sets the beginning-of-variables pointer to a spot about two-thirds of the way into Basic memory, leaving 22K for the common code and the longest overlay, and 16K for the variables, arrays and string storage.

If the common code plus the longest overlay is longer than about 24 disk blocks, you must adjust the pointer by increasing (on line 1030 of the boot program) the value poked into location 46. If your program produces an Out of Memory error when it runs, the pointer must be adjusted downward to increase the amount of space available for variable storage. (Out of Memory errors can also occur if your program uses too many nested For loops or too many nested GOSUBs, but this sort of error is not related to this pointer setting.)

Ideally, the pointer should be set to 8 plus the size of the common code plus the size of the longest overlay (code sizes are in disk blocks). This reserves the right amount of Basic space for the program text while leaving the maximum amount of space free for variables. [E]

# Mega-Magic

By Robert Bixby

## **RUN** It Right

*C-64*

Scroller is a brief machine language routine that turns your display into a continuous band running in either direction across the screen. Run Scroller Basic. It automatically saves to disk a machine language program file called SCROLLERMLA.

To access SCROLLERMLA, type in SYS828. The entire screen will move one column to the left. Type in SYS892 and it'll shoot back to where it started. You can scroll any text screen endlessly this way, using a continuous loop such as: 10 SYS892: GOTO10. If you write a program with a series of Data statements to be printed vertically on the left margin of the screen, SCROLLERMLA will scroll a message as long as the memory available in your computer.

Random Scroller demonstrates a potentially valuable feature of this scrolling program. By typing lines 30-70 into your program (perhaps as a subroutine) and setting the variables to appropriate values, you can scroll any section of the screen in either direction. Set T equal to the number (0-23) of the top screen line of the section to be scrolled, B to the number (1-24) of the bottom line of the section, R to the right margin and L to the left margin.

Load and run Random Scroller to see what I mean. Enter your own values for T, B, R and L. See if you can scroll different sections of the display in opposite directions at the same time. Experiment and have fun. One note of caution, however: Scrolling lines beyond line 24 will certainly Osterize your Basic program, and setting T to a value greater than B is likely to crash the computer. ☐

# Halloween Story

By W. O. Nelson

## **RUN** It Right

*C-64*

This Halloween program will delight all the children in your family and your neighborhood. It starts with an introductory sequence, including a background story printed on the screen, little sprite pumpkins that pop up from behind vines and some Halloween music. Then the "great pumpkin" is revealed, hanging bright orange in the black night sky. Every few seconds a sprite witch flies by, and once in a while lightning flashes and thunder rolls.

The pumpkin has black eyes with white sprite pupils that follow the witch across the screen, and a black nose and mouth. Curved vertical lines run up its sides, and a line circles the top where the lid should be. The pumpkin is stored in a picture file called Pumpkin, which the main part of the program loads to display the image on the screen.

To use the program, type `LOAD "HALLOWEEN".8`. The program is short, but cycles continuously, so you can let it run all evening if you like. Last year we put our monitor in the window so trick-or-treaters could see it. Believe me, we were the talk of the neighborhood! ☐



## Please send me back issues of ReRUN

- ☐ Gamepak ☐ Cassette version(s) at \$11.47\*  
☐ Summer Edition ☐ Disk version(s) at \$21.47\*  
☐ Fall Edition  
☐ Winter Edition  
☐ January/February 1986 \*\*  
☐ March/April 1986  
☐ May/June 1986  
☐ July/August 1986  
☐ Productivity Pak II

\* Prices include postage and handling. For foreign air mail, please add U.S. \$1.50 per item and \$25 per subscription. Prepayment only.

\*\* All 1986 editions contain 128-mode programs and are available on disk only.

☐ Payment Enclosed ☐ MC ☐ VISA ☐ AE

Card # Exp. Date  
Name  
Address  
City State Zip  
Signature

ReRUN • 80 Elm Street • Peterborough, NH • 03458

## BEAT THE RUSH!

Please send me:

- ☐ 1 year (6 issues) for \$89.97  
☐ November/December ReRUN disk for \$21.47.\*

\*Available in December.

Includes programs for C-64 and C-128 (in both 64 and 128 modes).

Price includes postage and handling. For foreign air mail, please add U.S. \$1.50 per item and \$25 per subscription. Prepayment only.

☐ Payment Enclosed ☐ MC ☐ VISA ☐ AE

Card # Exp. Date  
Name  
Address  
City State Zip  
Signature

ReRUN • 80 Elm Street • Peterborough, NH • 03458

# ***14 RUN Programs Included on this Disk:***

*Personal Finance   Turtle Graphics  
Education   Programming Aids  
Small Business Applications*

## ***From the September RUN:***

- ▶ The Loan Arranger
- ▶ Turtle-Tutor for Tykes
- ▶ RUN Basic
- ▶ Programmers, Take Note!
- ▶ Sign Maker
- ▶ Instant Data Statements

## ***From the October RUN:***

- ▶ 64 Personal Ledger
- ▶ High-Resolution Revolution
- ▶ As the Word Turns
- ▶ Making a Pointer
- ▶ Double Vision
- ▶ Chain Your Programs
- ▶ Mega-Magic

## ***PLUS Bonus Program:***

- ▶ Halloween Story

If any manufacturing defect becomes apparent, the defective disk will be replaced free of charge if returned by prepaid mail within 30 days of purchase. Send it, with a letter specifying the defect, to:

**ReRUN • 80 Elm Street • Peterborough, NH 03458**

Replacements will not be made if the disk has been altered, repaired or misused through negligence, or if it shows signs of excessive wear or is damaged by equipment.

The programs in ReRUN are taken directly from listings prepared to accompany articles in *RUN* magazine. They will not run under all system configurations. Use the RUN It Right information included with each article as your guide.

The entire contents are copyrighted 1986 by CW Communications/Peterborough. Unauthorized duplication is a violation of applicable laws.

©Copyright 1986 CW Communications/Peterborough



**CW COMMUNICATIONS/PETERBOROUGH**